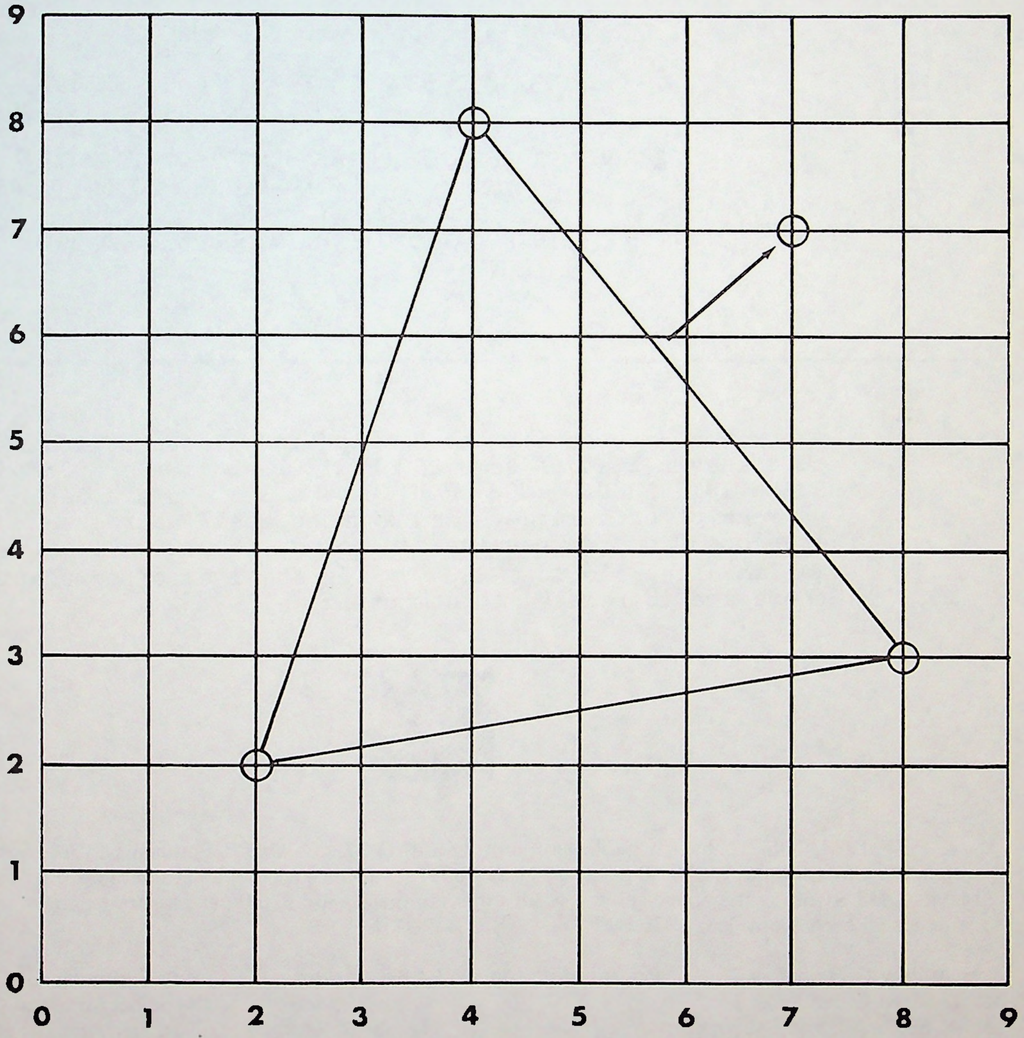
$2.50 45

Popular Computing

December 1976 Volume 4 Number 12



GENERATING TRIANGLES — CONTEST 14

--see page 3

N-SERIES 45

Log 45	1.6532125137753436793763169117857375916320678469 19283
ln 45	3.8066624897703197573912498070712390488205824699 14017
$\sqrt{45}$	6.7082039324993690892275210061938287063218550788 34577
$\sqrt[3]{45}$	3.5568933044900628060061546222389115849780869144 99437
$\sqrt[10]{45}$	1.4632591596632237867670171093417867254300480254 36069
$\sqrt[100]{45}$	1.0388004405460806648916652631282632901709996158 80928
e^{45}	34934271057485095348.03479723340609953341165649751815 4260126089844446933178520866 7171
π^{45}	23536629513782926058032.4894997597310579763169499863 6494464180570960330846878055 02
$\tan^{-1} 45$	1.5485777614681775602912076752295920085549668727 38475

The Science for the People Study Group periodically publishes a list of some 200 PERIODICALS THAT PROGRESSIVE SCIENTISTS SHOULD KNOW ABOUT. The list is available free on receipt of a large-sized stamped, self-addressed envelope from Progressive Technology, Box 20049, Tallahassee, Florida 32304. In the list of newsletters, there are 18 related to computing.

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 to the above rates. For all other countries, add $3.50 per year to the above rates. Back issues $2.50 each. Copyright 1976 by POPULAR COMPUTING.

Editor: Audrey Gruenberger
Publisher: Fred Gruenberger
Associate Editors: David Babcock, Irwin Greenwald
Contributing editors: Richard Andree, William C. McGee, Thomas R. Parkin
Advertising Manager. Ken W. Sims
Art Director: John G. Scott
Business Manager: Ben Moore

GENERATING TRIANGLES--A Contest

On a grid of 100 points, each point is located by two coordinates. Taking these coordinates as 2-digit numbers, their squares (taken modulo 100) are also 2-digit numbers.

Any three points that are non-collinear form a triangle on the lattice. The sum of the squares of the coordinates locate a new point on the lattice. This fourth point can then be:

a) Within the generating triangle
b) Outside the generating triangle
c) On the generating triangle

For example, the points 22, 48, and 83 (shown on the cover) locate the point 77, outside the generating triangle. The points 33, 78, and 80 locate 73, inside the triangle. The points 32, 82, and 48 locate 52, on the triangle.

There are 161,700 ways to pick three points out of 100 (although not all of these combinations form a triangle). What portion of these triangles determine fourth points of classes a, b, and c?

Write a program to generate all possible triples of three points and tabulate the results as follows:

a) A triangle is formed for which the fourth point lies <u>inside</u> the triangle.
b) A triangle is formed for which the fourth point lies <u>outside</u> the triangle.
c) A triangle is formed for which the fourth point lies <u>on</u> the triangle.
d) The triple does not form a triangle.

The program may be written in any language (even APL). Entries will be judged on the following criteria:

1. Correctness of the results.

2. Elegance of the program.

3. Readability, comprehensibility, and maintainability of the program.

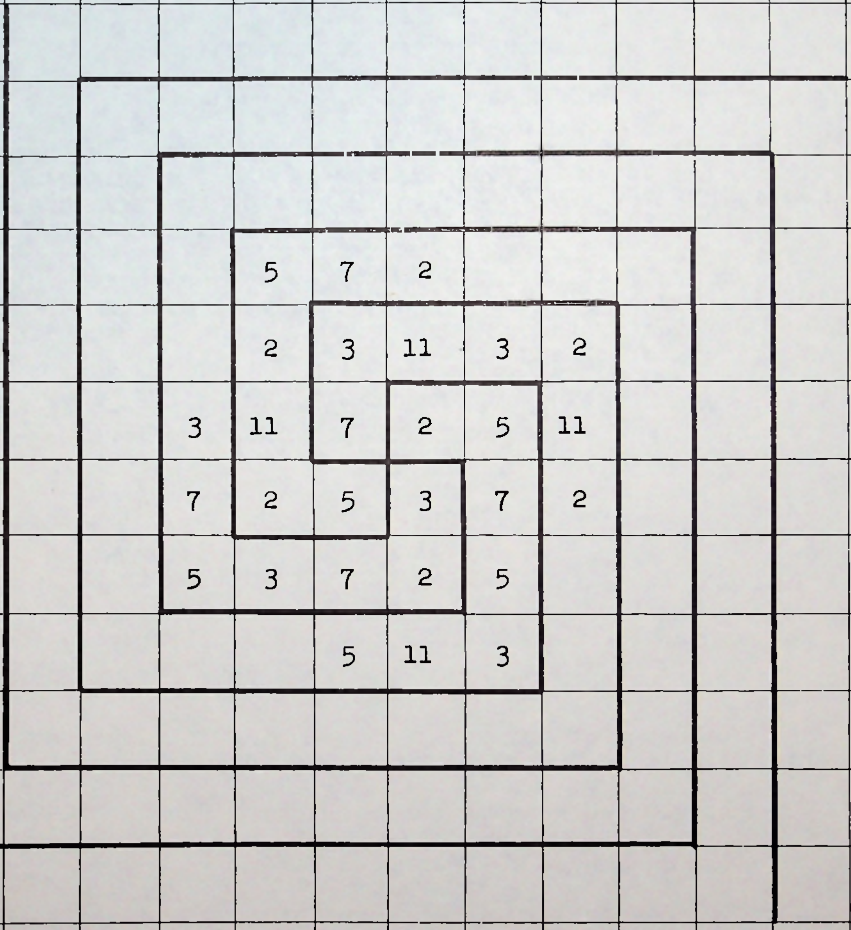
The opinions of the team of judges will be final; there will be only one winning program.

Contest entries must be received by February 28, 1977 at POPULAR COMPUTING, Box 272, Calabasas, California 91302.

PC45-4

gcd

In the pattern shown here, the number in each cell has no factor in common with the numbers in the eight cells bordering it. The numbers have been inserted into the pattern in order, starting with 2, 3, 5, and 7 at the center. Then each of the four paths (taken in rotation: path 2, path 3, path 5, path 7, path 2, and so on) is extended one cell, with the contents always the smallest possible integer greater than one.



PROBLEM 146

The path is to be extended, and the sequence along path 2 is to be listed.

The Circulating Pump

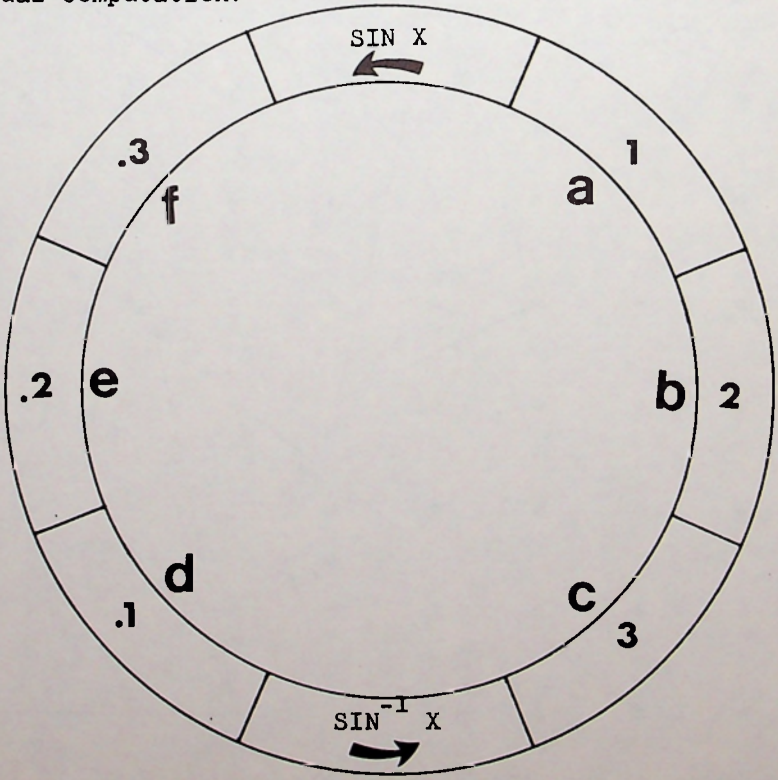
Cells a, b, c, d, e, and f contain the numbers 1, 2, 3, .1, .2, and .3 respectively to start. The numbers are circulated, one cell at a time, so that the number in cell a moves to cell f, the number in cell b moves to cell a, and so on. All six numbers move at once.

In passing from a to f, the sine is taken. In passing from d to c, the arcsine is taken <u>and summed</u>.

It is assumed that the sine can be taken of any number of radians, and that the arcsine will always be the principle value; that is, the arcsine will be an angle x such that x is less than or equal to $\pi/2$ and greater than or equal to $-\pi/2$.

Six moves will produce one complete revolution. The Problem is, what will be the contents of the cells after 100 revolutions? After 1000 revolutions? Cells d, e, and f will always contain numbers that are less than or equal to <u>one</u> in absolute value. Can the contents of cells a, b, and c increase without bound?

Can any of these questions be answered without actual computation?



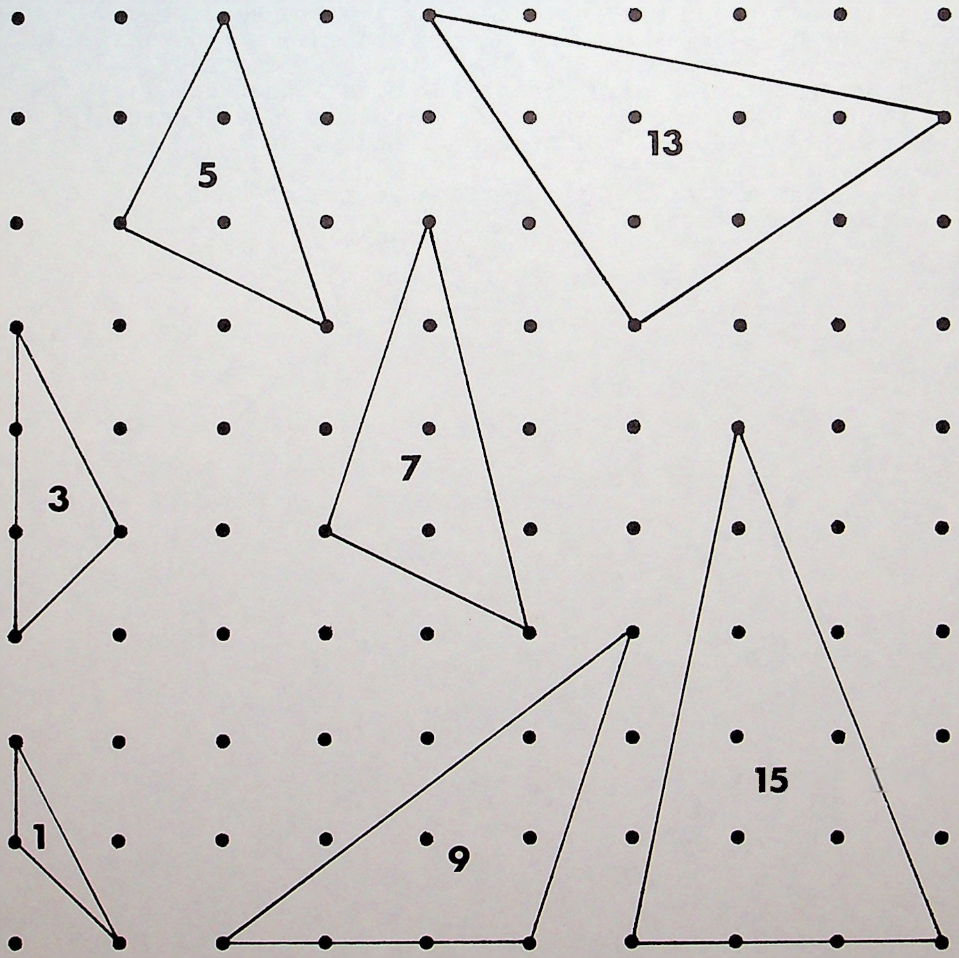
PC45-6

Wendy's Problem

On a 10 x 10 square array of points, there are 161,700 possible combinations of three points. Most of these combinations form triangles, and these triangles (taking the lattice size as one unit between points) can have areas ranging from 1/2 to 81/2 square units. The figure shows some triangles, with twice the area given in each triangle.

A program is to be written to run through the 161,700 combinations of three points, to produce a listing of how many do not form a triangle and how many form triangles of area 1/2, 2/2, 3/2,...,81/2 square units.

As part of the solution, there should be clear indications of the changes to be made to the program when the main parameter changes from 10 x 10 to 11 x 11, to 12 x 12, and so on.



PROBLEM **148**

PRIME PATH

Among the prime numbers, 2 and 3 are exceptions to the rule that all primes are of one of two forms:

6K+1
or 6K-1.

In the pattern shown here, the successive odd integers are laid out in a path of square cells that turns at every prime:

turn LEFT for primes of the form 6K-1;

turn RIGHT for primes of the form 6K+1.

```
      97  95  93  91  89
                      87
                      85
              79  81  83
              77
              75
              73  71
                  69          59  57  55  53
                  67  65  63  61          51
                                          49
                                  43  45  47
                          37  39  41
                          35
                          33
                          31  29
                              27
                              25
                      19  21  23
              13  15  17
       7   9  11
   3   5
```

Eventually the path will cross itself, so that the cell containing 111 will also contain 147. Similarly, one cell will contain both 91 and 179. We seek a list of the contents of those cells containing more than one number, arranged in the order of the smallest number in the cell.

A Problem "for undergraduate sophomore mathematics"

Contributing editor Professor Richard Andree offers this problem:

Show that the series:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{6} + \frac{1}{8} + \frac{1}{9} + \frac{1}{12} + \cdots$$

whose terms are the reciprocals of those positive integers which are divisible by no prime greater than 3 converges to an integer.

This is a variation of a problem first presented in our issue No. 10:

Prepare a list of all integers that contain only the factors 2, 3, and/or 5.

Three different solutions to that problem were given in issue No. 16, in the third article of our Art of Computing series. In Andree's problem, the series involves the integers that contain only the factors 2 and/or 3. The series:

$$\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots$$

(taken from the 2-3-5 problem) should also converge.

To what values do these two series converge?

A Pocket Calculator Problem

Suppose that your pocket calculator handles trig functions, and is in radian mode. You turn it on and depress the COS key repeatedly. After 48 depressions, the display has converged, to 8 digits (60 depressions for a 10-digit machine), to .7390851332.

You have just solved a rather high powered equation with just one key; namely, a root of

$$\cos x - x = 0.$$

Other trigonometric equations can be solved by balancing two or more functions. For example, you can solve for a root of

$$\sin x + \cos x - \tan x = .5$$

Pick a value of x (say, .73 radians), evaluate the left side of the equation, and then:

if the sum is greater than .5, make x larger
if the sum is less than .5, make x smaller

and thus zero in on the true value (around .7398).

There are many equations that are readily solved by such cut-and-try methods with a pocket calculator:

(A) $\cos x - \tan x = 0$

(B) $\sin^2 x - \cos^2 x + \tan x = 0$

(C) $\cos^2 x + \sin x + \tan x = 2$

(D) $\tan^2 x - \sin x - \cos x = 0$

(Along the way, we seemed to have also solved the equation

$$\sin x - \tan x + x - .5 = 0$$

--or did you notice that?)

Four-Square Ratio

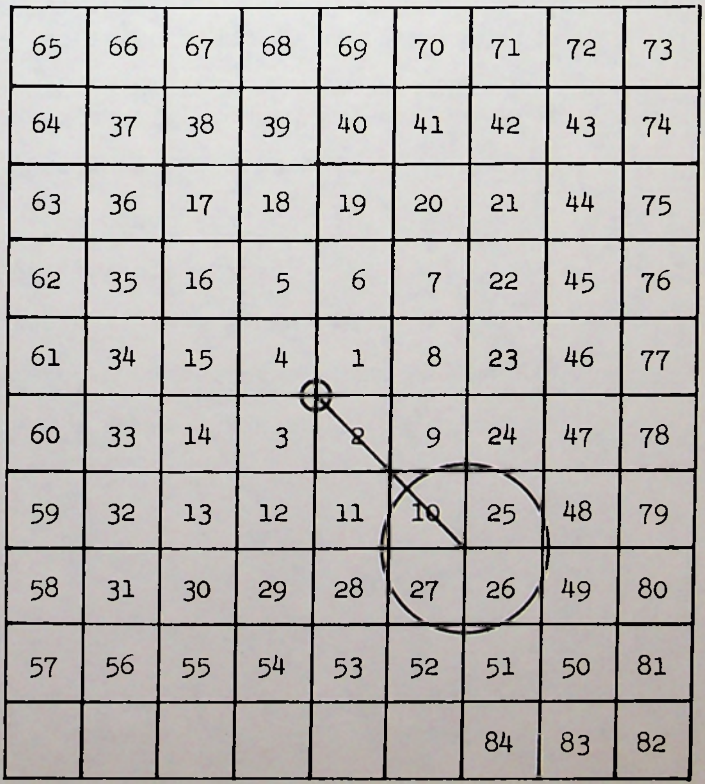
On an infinite grid of squares, each square is assigned a value, in a spiral around the origin, as shown in the diagram.

For each set of four contiguous squares, such as the four that are circled, the following is calculated: the sum of the four values divided by the (squared) distance from the origin. For the four squares shown in the diagram, this gives 88/8 = 11.

Except for the four squares surrounding the origin, this ratio can be calculated for every other lattice point. If it is done in the order in which the squares are numbered, successive new larger and smaller values of the ratio are recorded, to make a list that begins:

new larger	new smaller
16.0	16.0
20.0	11.0
28.0	10.4
36.0	8.75

Problem: extend the calculations, to show either that the ratio has reached its maximum and minimum values, or to find new maximum and minimum values.



65	66	67	68	69	70	71	72	73
64	37	38	39	40	41	42	43	74
63	36	17	18	19	20	21	44	75
62	35	16	5	6	7	22	45	76
61	34	15	4	1	8	23	46	77
60	33	14	3	2	9	24	47	78
59	32	13	12	11	10	25	48	79
58	31	30	29	28	27	26	49	80
57	56	55	54	53	52	51	50	81
						84	83	82

Braiding

A
PROJECT
FOR
A
PROGRAMMING
CLASS

1	4		← Contents
1	4	7	← Address
2	5		← Contents
2	5	88	← Address
3	6		← Contents
3	6	9	← Address

Given nine words of storage, addressed as shown. Words 1 through 6 contain the values 1 through 6 to start.

The average of the contents of words 1 and 4 is to be stored in word 7.

The geometric mean (i.e., the square root of the product) of the contents of words 2 and 5 is to be stored in word 8.

The sum of the contents of words 3 and 6 is to be stored in word 9.

The situation at this stage is:

1	4	2.50000
1	4	7
2	5	3.16228
2	5	8
3	6	9.00000
3	6	9

The array is now to be shifted to the left, in order to repeat the process. However, the words are to be braided, as follows:

Move the top row straight across; that is, $(4) \longrightarrow (1)$, and $(7) \longrightarrow (4)$.

Criss-cross the bottom two rows; that is, $(5) \longrightarrow (3)$, $(6) \longrightarrow (2)$, $(8) \longrightarrow (6)$, and $(9) \longrightarrow (5)$.

After the next arithmetic operations, we have:

4	2.50000	3.25000
1	4	7
6	9.00000	7.3485
2	5	8
5	3.16228	8.16230
3	6	9

After this (and all subsequent even stages), the bottom row is moved straight across and the top two rows are crossed. After the next arithmetic operations we have:

9.00000	7.34850	8.1742
1	4	7
2.50000	3.25000	2.85040
2	5	8
3.1623	8.1623	11.3250
3	6	9

The braiding process now reverts to the odd-numbered stage; namely, move the top row straight across and criss-cross the bottom two rows.

We record the successive contents of word 8:

1.	3.16228
2.	7.34847
3.	2.85044
4.	9.61427
5.	7.96512
6.	7.38947
7.	10.25340

and so on.

(For checking purposes, the following values are known:

50.	16747.497
100.	49268164.
200.	5.22426 E14
300	2.72194 E21
400	4.79420 E28
500	5.08364 E35)

A program is to be written to perform this task; this should be simple and straightforward in almost any language.

The class should now examine all the programs and rate them (say, on a 5-point scale) on the following features:

A) Readability. (Is the code itself clear? Are the comments clear and pertinent?)

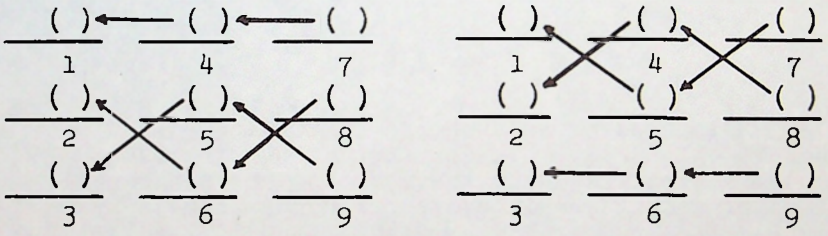
B) Comprehensibility. (Can you follow what is being done on every line?)

C) Maintainability. (If the logic is to be changed, is it clear how such changes could be made?)

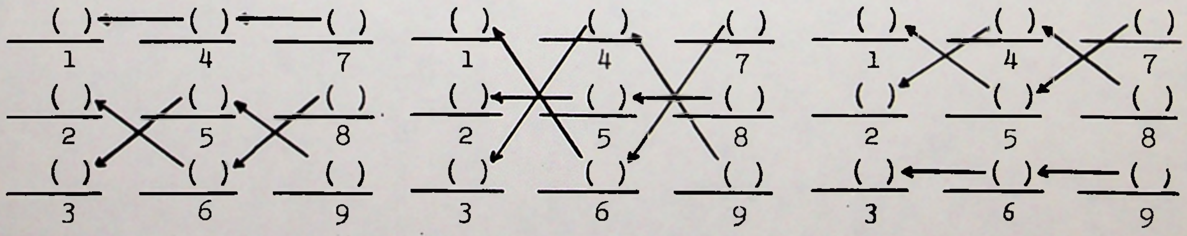
D) Efficiency. (Will the program use excessive compile time? Excessive execution time?)

Each student should now take someone else's program (the listing and the input deck) and make the following modification:

The original scheme called for 2-way braiding:



But now the braiding should proceed on a cycle of three:



and the contents of cell 8 should go as follows:

1.	3.16228
2.	7.34847
3.	8.13242
4.	6.79831
5.	7.19375
6.	8.04635
7.	10.95394
8.	19.24014
9.	21.31486
10.	18.01590

The new programs should be rated by the class on the same criteria as before.

During the first phase, the class should not be told about the following phase.

The 3X+1 problem has appeared in POPULAR COMPUTING in issues 1, 4, and 13. We will not bother to restate the problem. The following table shows the appearance of strings of successive integers having the same A value (the count of the number of terms generated in the 3X+1 process). For example, the 5 consecutive integers 98, 99, 100, 101, and 102 all go out in 26 terms. The asterisk indicates the first appearance of a string of that length.

3X+1
Strings
--again

2	13*	19	671470	36	1264959
3	30*	20	269717	37	
4	317*	21	152216	38	
5	102*	22	212181	39	2754407
6	391*	23	578013	40	596349*
7	949*	24	221208	41	2886393
8	1501*	25	57370*	42	
9	1688*	26	393242	43	
10	139103*	27	252574*	44	
11	258229	28	1510806	45	
12	666523	29	331806*	46	
13	961101	30	524318*	47	3247193*
14	3000*	31	1088159	48	
15	262159	32	913350	49	
16	262967	33	2512873	50	
17	7099*	34	1541342	51	
18		35	1032909	52	3264480*

symposium 18

List of attendees for the June 6, 1976 symposium

PAUL ARMER	Center for Advanced Studies in Behavioral Science
ROBERT BEMER	Honeywell Information Systems
BARRY GORDON	IBM
IRWIN GREENWALD	Xerox Data Systems
FRED GRUENBERGER	California State University, Northridge
DANIEL McCRACKEN	Author
CLARENCE POLAND	IBM
RICHARD TANAKA	California Computer Products
ROBERT WHITE	Informatics
JOSEPH WEIZENBAUM	Laboratory for Computer Science

Every year since 1958, a one-day invitational discussion session has been held to discuss current topics in computing. Paul Armer and Fred Gruenberger have co-hosted these affairs, which were sponsored until 1968 by the Rand Corporation, and since then by California State University, Northridge.

The 1976 session will probably be the last of these symposia, on the grounds that they have outlived their usefulness. Each year, the attempt was made to gather a homogeneous group of experts, and this tended toward the senior citizens of the field. There is some evidence that the results were fruitful, but that the impact, if any, has attenuated over the years. Like any good vaudeville act, it seems wise to get off the stage while the audience might still be enjoying the show.

It seemed like a good idea, therefore, to change the theme for 1976; to avoid the burning questions of the day and consider what are likely to be controversial topics of 2006. In the announcements of the meeting, the thought was offered that it was not likely that in 2006 the computing fraternity would still be debating the relative merits of Fortran vs. PL/1. On the other hand, it would still be true that one shouldn't calculate constants inside the loop. The questions to be discussed, then, are the things that might endure. Of course, as you might expect, several of you disagreed with me on the things that I listed that we wouldn't be debating 30 years from now.

GORDON: I didn't know that we were debating Fortran vs. PL/1 now.

GRUENBERGER: You can get any ten computer people to spend hours hotly debating that one.

BEMER: We should clear up our terms of reference. The physical evolution in computing is visible and tremendous, and we can reasonably expect more of it in the next 30 years. But look at the people in computing. There I don't see any evolution, or much improvement. People will learn one language, like COBOL, and they are willing to live with that for the rest of their life. Tony Pizzarello gave some talks at our place on structured programming and P-notation languages and things like that, and found that only about 10% of the people could be retrained at all. They don't seem to care. They will, indeed, argue the merits of COBOL, PL/1 and Fortran for years.

POLAND: But with any luck, those people will die off soon enough and a new breed will emerge.

WHITE: Except that the old ones propagate. Fortunately, COBOL is not hereditary.

WEIZENBAUM: My town, Concord, Massachusetts, decided to install a computer system to determine real estate taxes, taking into account many more "variables" than could previously be used, and apply sophisticated techniques like linear programming. I think it's a terrible mistake, inasmuch as events in this system (unlike process control) do not take place every few milliseconds, and hand methods would do the job nicely. But the pitch of the firm that sold the system to my town is that the method of determining the tax will be so much better, since it will invoke magic like factor analysis, so that the causative factors can be determined. When this system becomes operational, someone will ask how his tax was determined, and the tax assessor will have to say "I don't know." This is another kind of dependency, much different from the kind that requires you to know how a compiler works. The user will not know what theory is being applied, and will certainly not know the algorithm that has been implemented. It seems to me that the town will either have to hire someone who knows and understands the algorithms and can explain them to the citizens, or it will have to revert to the old hand methods. This strikes me as typical of the thing we're rapidly getting into. The abdication of responsibility is not a consequence of the technical incomprehensibility of the system, but is a consequence of the substantive incomprehensibility of the system. The real point is that there is no theory in such things. Someone made a system that worked, and then other things were patched on to it. It was generated in performance mode, and then the general manager asked "Can we also do that?" and the system grew. Even with the best will in the world, it is no longer possible to find out the basis on which decisions are being made.

GORDON: At one time computers were a scarce resource and were very expensive. Perhaps it made sense then to tailor the problem solution to the available equipment, but it doesn't make sense today.

WHITE: It's the only thing that can be measured. You can count usage factors, but you can't measure utility and quality.

GREENWALD: Let's not criticize the priesthood too much on this issue. The reason that microsecond chasing still goes on is benchmarks. Every vendor has to do them against other vendors, and I don't see that changing.

POLAND: And you don't see benchmarks set up to time the installation and measure the maintainability of a new job.

GREENWALD: In competitive situations, we learned that the company that gets there first will probably win, because that company gears the benchmarks to its system. I agree that 30 years from now we won't worry about calculating constants inside the loops; we won't be able to afford to and still do the things we've been talking about. On the other hand, if there is still competition, we'll still be counting machine cycles.

WHITE: It is also unlikely that there will be a change in the attitude of the people who run the systems that the purpose of the man at the far end is to feed information to the computer, rather than having the computer furnish useful information to him.

GRUENBERGER: Did you say, Clarence, that one day the property owner will be able to come to the tax assessor and find out the algorithm that was used to calculate his tax?

POLAND: Yes; that's a desirable objective. They used to be able to, and they should be able to with computers.

GRUENBERGER: You're dead wrong. They'll be able to hide behind that computerized system, and they will.

WHITE: That's a characteristic of bureaucracy, not of computing.

GRUENBERGER: Sure it is, but the computer makes it so much easier. We see it already. Department stores, banks, motor vehicle departments--they have all learned very well that no one argues when they calmly say "The computer says so." Not only won't they give you the algorithm, they won't be able to, because after two revisions of the program, no one will know what the algorithm really is any more, as Joe pointed out.

GORDON: What's more, the new system was sold on just that basis. The salesman told them that they could implement what they did before, and also lots of new things. Again, few people ever then ask "Should we do these things?" They are done simply because they can be done.

WEIZENBAUM: The firm sold the town a system that included all sorts of sophisticated things like factor analysis. The town wasn't even talking to a programmer, trying to explain how they assess taxes. But there is also a political context to be considered, with all the federal rules that towns must observe. The tax assessor would like very much to be able to escape the responsibility of making very tough decisions that are bound to make one segment of the community or another unhappy. The salesman from the computer firm offered him a way to avoid such responsibilities. The pitch was that the hard decisions would be made automatically, logically, scientifically and--best of all--"by computer." In buying it, the assessor has said, in effect, "and no one will be able to argue with me."

GRUENBERGER: Think of the 17 years or so of experience that has built up with banks using computers. They were one of the first groups to adopt widespread use. After all these years, if your bank slips $27 out of your account and you complain, you'll hear immediately "the computer did it." You can ask for the algorithm, but it will be a fight. All the way up to the level of the manager of a branch you'll only get the same idiotic reply. You may eventually get your $27 back, if you squawk loud enough, and then you'll find that the computer did that. You are not supposed to argue with these pronouncements. We're here already.

BEMER: The hope is that IBM will see the handwriting on the wall . . .

GORDON: Nonsense. The handwriting on the wall is in hexadecimal, which is the worst crime ever perpetrated on the users. Talk about kowtowing to the machine! A decimal chip costs no more than a hex chip. It's an outrage to the users, and as long as it exists, we have a nerve talking about doing anything to help them.

GRUENBERGER: How do they do it in other industries? It's certainly true that there is always someone around to sell you junk, but in other industries you can buy quality if you want to.

GORDON: You don't buy a car to drive it at 90 miles per hour all day. You expect it to be idle most of the time. Yet in our industry it's a crime to have the WAIT light come on. Where did that get started?

GRUENBERGER: It's a matter of scale. 747's don't sit idle, either, but Piper Cubs do. Similarly, big 370's are seldom idle, but IBM 5100's will be idle most of the time. It's a matter of how much you have to invest just to get the thing there at all.

WEIZENBAUM: And even if you don't drive the car 90 MPH all day, they make it to do just that--and you pay for it.

POLAND: Even before computers, a bookkeeper knew how he got his results (whether they were right or wrong). One of his fundamental rules was: never erase. He would either correct, with a line drawn through an entry, or he would make a reverse entry. Moreover, he retained all the figures that led to his final balance.

In computing, we threw out this fundamental piece of folklore, not recognizing that it was an excellent idea. It is only just recently that we are getting around to restoring it.

GREENWALD: The places where we are now doing things considerably better are the places where we were forced by user requirements. The world of data processing (as opposed to the world of computing) may force us to do more of the things we've been talking about. Part of our troubles stem from the fact that us senior citizens generally came out of the area of scientific computing.

WEIZENBAUM: We may also be in the same position as the generals who figure out how the last war should have been won. We all grew up with large, expensive machines. Maybe the impact of the new little machines will alter people's thinking.

We started out here saying that 30 years from now people would no longer be debating Fortran vs. PL/1. Perhaps they will; perhaps there is no way of backing out even of that. The disaster we've already experienced in this connection is the language BASIC, and that is now irreversible.

GRUENBERGER: I doubt that we could have something as disastrous as a complete breakdown of EFTS, but what I think is likely is something like having the credit rating of all the veteranarians in the country wiped out one day.

WHITE: No, there could well be a complete breakdown, through a chain reaction, much like the chain reaction that took out the northeast power grid a few years back.

GORDON: And that could get worse, too. The power grid is now growing national, so that when the next blackout occurs, you guys out west can participate along with the rest of us.

WHITE: They patched up the local troubles by going national, in other words.

GORDON: We have a large telephone system, and we've never had that kind of national failure.

GRUENBERGER: No, it just has local failure continuously.

TANAKA: Suppose we carry this through, and actually get figures for the chance of disaster and its consequent cost, for any given situation. How is a manager supposed to use that information? We would tell him that for N dollars you can get so much protection and for 10N dollars you get so much more, and so on. There is no point unless you state the situation in terms of choices that the manager can make. At some point, the cost would put the company out of business right away.

BEMER: I feel that all the software fixes you'll ever need cost less than going out of business.

GREENWALD: But there is no real assurance that, for extra money, there will be any real improvement. We've attacked this problem from both ends. We started by asking how you define quality, how do you measure it, and what does it buy you? Then we shifted to the user's need for quality, and how do we sell him on buying it? We then have to prove our case, and for that we're back to defining quality and measuring it--we're in a vicious circle.

GRUENBERGER: It's like selling burglary insurance; what you need, at regular intervals, is some good burglaries.

There seems to be a passion for assigning numbers to things, as though that nails down some basic truth. Would word counts or other mumbo-jumbo establish that Dan writes better books than other people? Quite apart from the technical content, they are recognizable as well-written books. Do we have records of brush-strokes/hour for Rembrandt and Vermeer? Would such data help us in any way to establish that the paintings by those men are better, in any sense, than those of a thousand other painters? My point is that there are means available for certifying a program as good (comprehensible, maintainable, portable, modifyable-whatever you please) without having to attach numbers to it. Of course you can count the GOTOs, but such schemes will only reveal the extreme cases in either direction; they cannot discriminate in exactly those cases where you need a discriminator.

GREENWALD: It's hard to predict the future. We came out with an operating system in 1967 that was aimed primarily at the scientific time-sharing user. We are now running commercial transaction processing on it. That shift automatically makes that software a real kludge. It's 9 years old, and it has evolved, twisted, and turned. Its parameters were all wrong, because we couldn't foresee the future. I think that't true of more programs than people are willing to admit, and particularly in the applications area. (We noted earlier that vendor software tends to be better, on the whole.) I agree that better software out of IBM would have a large impact.

GRUENBERGER: We seem to bounce up and down like a yo-yo. Are things going to get better, or are they going to get worse?

POLAND: Both, of course. There will continue to be microsecond counters and bit chasers and other such experts (the typical output of universities) who are chasing that level of excellence. The vendors will have them, but the applications areas will have more of them.

GREENWALD: If the SILT report is correct, they will increase, just by the sheer numbers of new programmers.

WHITE: Sort of a Gresham's Law effect.

POLAND: On the other hand, we will see concurrently facilities that will permit those people to do their thing (and take the consequences) but will also permit the construction of computing systems wherein bit-chasing is not permitted to happen. These will be end-users-oriented facilities. I foresee a real dichotomy, around the year 2000, much worse than what we have now. Some of us can see it now; I think the split will be clear to all, long before 2000.

WHITE: In building a data base management system, I can envision having hardware do such things as "Give me a record with this key value."

McCRACKEN: OK, I can see such things coming. But take Irwin's PL/1 example. Carrying it to an extreme, we would get a machine having PL/1 as its machine language. That would be a disaster! We don't know how to design languages well enough.

GORDON: Precisely the point I made earlier; that's why language translators are still software. When we do know how to design languages, then it will move into the hardware.

POLAND: There are machines today whose internal language is APL--and so what? It doesn't do a dammed thing to help getting an application on the system. It has not helped to solve the software problem; the fact that APL is the machine's native language is an irrelevancy.

GORDON: We've seen it happen many times. Things that were explored in software (like emulation, or paging) until they were understood, then moved into the hardware. Ultimately, I can see JCL being cast into hardware. Even programming structures (like IFTHENELSE) could be put into hardware. Again, it's too early to do it, because we don't thoroughly understand these things yet. They aren't standardized and they haven't settled down. When they do, and we put them into the hardware, we can devote more attention to the applications.

McCRACKEN: If PL/1 were in hardware, it would be just as bad a language.

GORDON: Which may be why it is not in hardware.

GRUENBERGER: I said earlier that I thought we could agree that eventually all programs would have to be structured, and we didn't agree. I'd like to hear why not.

POLAND: The word is "programs." We will not be doing things by writing programs in the sense that we do it today. We will use non-procedural processes. For such processes (e.g., non-procedural conversations with a computing system) the whole concept of structured programming is irrelevant. The things we call programs will be produced, on the whole, by vendors and those will probably be structured. In the same vein, I believe that chips will be produced by design automation, and not by any process that puts pen to paper.

GRUENBERGER: Let me reword it, to find something we might agree on. "Sets of instructions for computers, if they are spaghetti-like, full of GOTOs, and a mishmash, shall be outlawed."

GREENWALD: But there will be languages (assembly, for example) that do not allow the constructs of structured programming, so it can't be.

WEIZENBAUM: Let me try it. We have verbal and written communication and some standards of literacy, so that there is a generally accepted level below which we can say "That won't do." Could we not have a minimum level of literacy in computer programs? This is independent of the fact that the programs may work. Maybe it should be called elegance, and the level could be low, but shouldn't there be some minimum level?

GREENWALD: To my way of thinking, you're now differentiating between structured programming and structured design. There is no reason why a program that is implemented in assembly language couldn't be written in structured language.

GORDON: There is no reason why hardware, within the next 5 years, cannot accept structured codes directly.

GRUENBERGER: Perhaps an example of Joe's literacy level might be the common student error of writing

LOAD ACCUMULATOR
LOAD ACCUMULATOR

in succession--that would be below the level of program literacy. We could find much better examples, I'm sure.

GORDON: But who decides and enforces this level? When the year-end report has to get out, is someone supposed to say "We can't run this because it's aesthetically displeasing?"

GRUENBERGER: Well, you do just that with the written report that goes with it; you don't let illiteracy creep in there.

WEIZENBAUM: In general, in business, a man being considered for promotion who is asked to give a talk to his peers or to customers, who turns out to be functionally illiterate, will probably be denied the promotion.

*: Not at IBM.

WEIZENBAUM: I'm not talking about an occasional lapse in grammar or misuse of words, but functional illiteracy. Now that computer languages have become so much a part of our means of general communication, I think we should work toward establishing modes of conduct in those languages as we have in English.

WEIZENBAUM: Two things have changed dramatically in the 18 years of these sessions: (1) hardware reliability is taken for granted, and (2) computers are no longer a scarce resource, and no longer expensive. That last fact may come as news to people who are spending a million dollars on an installation, but let's hope they are spending it appropriately and getting far more for their dollar than 18 years ago. At least we in the industry recognize that whatever scarce resources end-users have to wrestle with, it is no longer raw computing power. That fact has not taken effect in the outside world. The users are still sub-optimizing (that is, optimizing at too low a level), and we should work on that, but if there is one single message we should produce, it is that computer time is no longer crucial; it can be traded off against quality and reliability.

GRUENBERGER: But that message is already widespread among students, and their interpretation of it is "computing is so cheap, we can afford to do it sloppy." They believe devoutly that (a) computing power is a free good and (b) there is no point to using intelligence--we can overcome stupidity with sheer speed. There is nothing to be done about (a), because it is obviously true at most universities. The tragedy is that (b) is patently false, and they proceed to prove it, over and over, but still believe it. I haven't found a way to dispell that myth.

WEIZENBAUM: The first time I heard that sentiment (that the machine is so fast that I don't have to be bright) was from a programmer on the RAYDAC, around 1953. We should be careful, when we stress the low cost of computing today, that we don't re-sell that message. The real message is now we can afford to be as elegant as we wish, even if it costs you machine time.

GORDON: But even that is the wrong message. It is cheaper, overall, to do it right, right away. Compilers take minutes; loops take nanoseconds. If it never was true, it is still not true: CPU speed is never a substitute for good programming.

The material above is excerpted from
the complete transcript (56 pages) which
is available for $10 from the

Engineering Center
School of Engineering and Computer Science
California State University, Northridge
Northridge, California 91330